

(19)



Europäisches Patentamt  
European Patent Office  
Office européen des brevets



(11)

EP 1 014 265 A1

(12)

## EUROPEAN PATENT APPLICATION

(43) Date of publication:  
28.06.2000 Bulletin 2000/26

(51) Int Cl.7: G06F 11/00

(21) Application number: 99309818.5

(22) Date of filing: 07.12.1999

(84) Designated Contracting States:  
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU  
MC NL PT SE  
Designated Extension States:  
AL LT LV MK RO SI

(30) Priority: 15.12.1998 US 211967

(71) Applicant: LUCENT TECHNOLOGIES INC.  
Murray Hill, New Jersey 07974-0636 (US)

(72) Inventors:

- Holzmann, Gerard Joan  
Murray Hill, New Jersey 07974 (US)

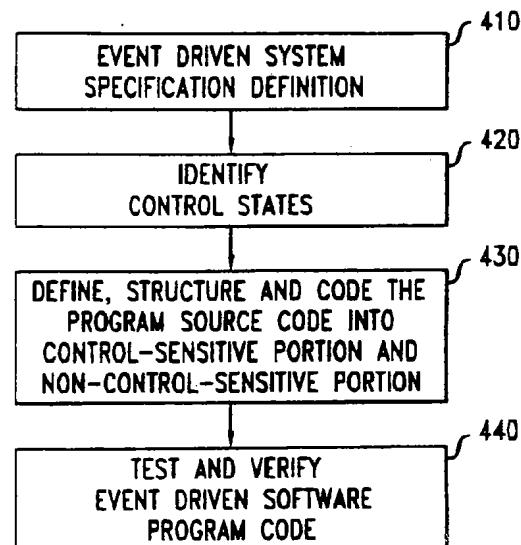
- Lane Thompson, Kenneth  
Watchung, New Jersey 07060 (US)
- Winterbottom, Philip Steven  
Watchung, New Jersey 07060 (US)

(74) Representative:  
Watts, Christopher Malcolm Kelway, Dr. et al  
Lucent Technologies (UK) Ltd,  
5 Mornington Road  
Woodford Green Essex, IG8 0TU (GB)

### (54) Method and apparatus for testing event driven software

(57) A technique for testing event driven software. In accordance with the technique, the source code of the event driven software is directly converted to a automaton based model useful in verifying that the program code complies with the desired properties defined by the user. More particularly, the event driven system program code is translated into a target language for a particular model checker. Such a translation results in a model which contains statements directed at whether execution of the program code will affect the behavior of the event driven system. Thus, this model extraction process can be used as input to a logic model checker for determining whether event driven system complies with the desired correctness properties specified by the user. Advantageously, the model extraction process and application of the model checker occurs in a direct and dynamic fashion from the subject event driven system program code without the need for user intervention.

FIG. 4



**Description****Field of the Invention**

5 [0001] The present invention relates to testing computer software programs and, more particularly, to a technique for testing event driven application programs.

**Background of the Invention**

10 [0002] In well-known event driven software systems (also referred to in the art as "reactive systems"), certain functionality occurs when an event is generated and communicated to an executing process performing particular actions based upon the occurrence of the particular event. More particularly, event driven software is used to define the behavior of a system, e.g., a computer, in response to events that are generated by other systems, machines or peripheral devices connected thereto. Well-known examples of event driven systems include: telecommunications applications, 15 call-processing software, data communications software, device drivers, and computer interface applications. As will be appreciated, these types of event driven systems are state oriented. That is, in each state of a system, one specific event from a predefined set of events is expected to occur, and the system is designed to respond to the event in a well-defined manner.

20 [0003] Briefly, as is well-known, a state machine converts a time series of event stimuli or input data through some function into a time series of responses or output data. For example, in an event driven system, at each point in time, each machine is in a particular so-called "control state" which defines the events that are expected to be generated by other machines, and the type of responses that are to be generated upon the arrival of each such event. Typically, the response includes the execution of a finite piece of program code, the generation of new events, and the identification of a new control state for the particular machine responding to the incoming events. Thus, the state machine 25 model of the event driven system plays a critical role in the development of the application program code by the programmer for the system.

25 [0004] Typically event driven application software is written in higher level general-purpose programming languages such as the well-known "C" or "C++" programming languages, or a special-purpose language such as the well-known ITU Systems Description Language ("SDL"). In developing the actual program code for such systems it is common to 30 describe the behavior of the relevant machines in terms of state machines. Using a higher level programming language, e.g., C, software programmers will write event driven software code for the particular application. Typically, the software program development begins by reviewing a so-called system specification which provides a logical description of the desired operations for the application. Working from the specification, one or more programmers will write the necessary code to implement the operations.

35 [0005] This conventional programming process results in an application program which contains no particular visual context amongst and between the various control states of the event driven system. That is, the program code can be viewed as one large "flat" file with a series of program instructions. Further, as the original specification changes during program development, e.g., additional functionality is added to the system, the requisite program code and additional control states are typically added to the source program in a somewhat arbitrary order. In order for a programmer to 40 understand the logical flow from control state-to-control state directly from the program source code it is necessary to manually trace the execution thread through the source, e.g., tracing branch instructions or "goto" statements. As will be appreciated, the tracing of such logical flow, particularly in very large source code programs, is a tedious and time consuming task. This also leads to certain complexity in the testing of the software for so-called "bugs" and to determine whether the application program implements the desired features.

45 [0006] More particularly, a critical state in the development of the application software code for such systems is the testing of the code to validate the properties of the software and to insure that the behavior of the system complies with the design objectives. There are a number of well-known methods for testing event driven software. For example, Boris Beizer, *Black-Box Testing: Techniques for Functional Testing of Software and Systems*, John Wiley and Sons, 1984, describes a well-known testing arrangement wherein a set of test goals is defined from which a limited set of 50 test sequences is deduced. Typically, the set of test sequences is defined through a manual process by the testing engineer through examination of the system design. The set of test sequences is applied to the event driven software under test and either a "pass" or "fail" is recorded as the testing result. This testing methodology is typically applied to general software testing and is well-known in the art to be prone to human error and incomplete. In particular, the number of possible executions of the event driven software under test is incredibly large. In contrast, the number of 55 tests that can be performed on the software with real-time executions is limited by certain practical constraints such as the minimum time required to perform a single test from the total number of tests. As such, it is typical that no more than a few hundred tests can be selected and applied in accordance with this methodology. This results in a very low confidence level in the testing coverage.

[0007] A second event driven software method known in the art is described, e.g., by D. Lee et al, *Principles and Methods of Testing Finite State Machines - A Survey*, *Proceedings of the IEEE*, Vol. 84, pp. 1090-1126, 1996. In accordance with this testing method, an abstract state machine model for the event driven software program is constructed. The state machine model typically captures only a high-level description of the program code thereby leaving out most of the program details. Thereafter, well-known test sequence generation algorithms (see, e.g., Lee, *supra*) can be applied to the state machine model from which a relatively small set of tests is derived and applied to the software under test. As with the above-described testing method, this technique relies heavily on the manual construction of the abstract model of the event driven software. Therefore, this technique is sensitive to certain errors caused by mismatches between the formal relation between the software under test and the abstract model to be used for deriving the applicable tests. In addition, changes to the event driven software require the development of new tests which proves cumbersome and time consuming.

[0008] In a third commonly used testing method, see, e.g., C. H. West, *Protocol Validation by Random State Exploration*, *Proc. 6<sup>th</sup> IFIPWG 6.1 Int. Workshop on Protocol Specification, Testing, and Verification*, North-Holland Publ., pp. 233-242, 1986, a random exploration of the full behavior of the event driven software is undertaken in order to establish the software's compliance with the desired properties. The amount of detail in the full software description generally precludes any attempt to perform an exhaustive exploration of the software for testing purposes. Therefore, well-known statistical algorithms are used to extract and analyze random samples of the software's behavior. However, as with the other above-described testing arrangements, one recognized drawback in this testing approach is that the amount of extraneous detail in the full software description prevents an efficient exhaustive exploration of the full behavior of the software for compliance with high-level properties.

[0009] Therefore, a need exists for a technique that mitigates the above-described problems in the art and improves the programming development and testing of event driven software.

### Summary of the Invention

[0010] The present invention provides a technique for testing event driven software directed to the efficient testing of event driven software and making it possible to track the validity of future changes to the program code, as part of the normal maintenance, extension and revision process to the source code. More particularly, in accordance with the preferred embodiment of the invention, event driven software code is specified and written in accordance with the principles of the co-pending, commonly assigned patent application, entitled "Method and Apparatus For Developing Event Driven Software", Application Serial No. , filed on event date herewith. That is, in accordance with the preferred embodiment, the source code program is defined and structured into control-sensitive and non-control-sensitive format. In accordance with the present invention, the event driven source code program is parsed to extract the control states defined in the source program, and to convert the source code into an intermediate state machine format. Thereafter, in accordance with the invention, the intermediate state machine format is converted into a automata-based format for modeling checking purposes.

[0011] In particular, in accordance with the preferred embodiment, the intermediate state machine format is converted into an annotated automaton model. Illustratively, the annotated automaton model is a verification program written in the input language of a specific model checker. Importantly, in accordance with the invention, each statement from the source code program is mapped onto an abstraction within the automaton model. The mapping is achieved using a translation map which is defined at the initial stage of the verification process. The map need not be revised thereafter unless new types of instructions are introduced, e.g., after revisions are made to repair programming faults, into the source program. The mapping, *inter alia*, dictates a fixed translation of the particular statement into the target language of the model checker. Thus, instructions that appear in the map are systematically converted into the automaton model wherever such instructions appear in the source code.

[0012] In addition to the above-described map, in accordance with the invention, a so-called environment model is defined which encapsulates a minimal set of assumptions that must be made about the particular operating environment in which the event driven application is executed. More particularly, as the model checker input language is provided to the model checker, the environmental model is applied. As a result, in accordance with the preferred embodiment of the invention, the verification of the properties of the event driven system is made subject to the environmental model during the testing and checking of the event driven software by the verification system. Thereafter, the testing and checking results are output, in a well-known manner, by the model checker for determining whether the event driven system conforms with the user's desired execution properties and behaviors.

[0013] Advantageously, in accordance with the invention, the model extraction process and application of the model checker occurs in a direct and dynamic fashion from the event driven system program code without the need for user intervention.

Brief Description of the Drawings**[0014]**

5 FIG. 1 is a block diagram of an illustrative event driven system;  
 FIG.'s 2 and 3 show an example of a conventional C source code program for an illustrative telecommunications  
 call processing application useful in the illustrative event driven system of FIG. 1;  
 FIG. 4 shows a flowchart of illustrative operations for software development in accordance with the principles of  
 the invention;  
 10 FIG.'s 5 and 6 show an illustrative C source code program, structured and defined in accordance with the software  
 development operations of FIG. 4, for the illustrative telecommunications call processing application of FIG.'s 2  
 and 3;  
 FIG.'s 7 and 8 show a translated source code program derived, in accordance with the principles of the invention,  
 15 from the illustrative C source code program of FIG.'s 5 and 6;  
 FIG. 9 shows a flowchart of illustrative operations for testing event driven software in accordance with the present  
 invention;  
 FIG. 10 shows intermediate state machine code derived from the illustrative C source code program of FIG.'s 5  
 and 6;  
 20 FIG. 11 shows an illustrative map for the event driven system defined by the illustrative C source code program  
 of FIG.'s 5 and 6;  
 FIG.'s 12 and 13 show model checker input language derived from applying the illustrative map of FIG. 11;  
 FIG. 14. shows an illustrative set of library functions for use with the model checker input language of FIG.'s 12  
 and 13;  
 25 FIG. 15 shows an illustrative environmental model for the telecommunications processing application described  
 in FIG.'s 5 and 6;  
 FIG. 16 shows an illustrative automaton model for a particular control state defined by the telecommunications  
 processing application of FIG.'s 5 and 6;  
 FIG. 17 shows an illustrative full automaton model of the complete telecommunications processing application  
 described in FIG.'s 5 and 6;  
 30 FIG. 18 shows an illustrative computer system 1800 useful in executing event driven software code which is spec-  
 ified and written in accordance with the present invention; and  
 FIG. 19 shows an illustrative error trace, generated in accordance with the principles of the invention, for the  
 telecommunications processing application described in FIG.'s 5 and 6.  
 35 Throughout this disclosure, unless otherwise noted, like elements, blocks, components or sections in the figures  
 are denoted by the same reference designations.

Detailed Description

40 **[0015]** A portion of the disclosure of this patent document contains material which is subject to copyright protection.  
 The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent  
 disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all other  
 rights with respect to the copyrighted works whatsoever.  
**[0016]** The present invention provides a technique for testing event driven software. In accordance with various  
 45 aspects of the invention, the event driven system program source code is directly converted to a automaton based  
 model useful in verifying that the program code complies with the desired properties defined by the user. More partic-  
 50 ularly, in accordance with a preferred embodiment of the invention, the event driven system program code is translated  
 into a target language for a particular model checker. Such a translation results in a model which contains statements  
 directed at whether execution of the program code will affect the behavior of the event driven system, e.g., the execution  
 of the program code causes new events to be generated. Thus, this model extraction process of the preferred embod-  
 55 iment of the invention can be used as input to a logic model checker for determining whether event driven system  
 complies with the desired correctness properties specified by the user. Advantageously, in accordance with the invention,  
 the model extraction process and application of the model checker occurs in a direct and dynamic fashion from  
 the subject event driven system program code without the need for user intervention.  
**[0017]** The present invention can be embodied in the form of methods and apparatuses for practicing those methods.  
 55 The invention can also be embodied in the form of program code embodied in tangible media, such as floppy diskettes,  
 CD-ROMs, hard drives, or any other machine-readable storage medium, wherein, when the program code is loaded  
 into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the invention.  
 The invention can also be embodied in the form of program code, for example, in a storage medium, loaded into and/

or executed by a machine, or transmitted over some transmission medium, such as over electrical wiring or cabling, through fiber optics, or via electromagnetic radiation, wherein, when the program code is loaded into and executed by a machine, such as a computer, the machine becomes an apparatus for practicing the invention. When implemented on a general-purpose processor, the program code segments combine with the processor to provide a unique device that operates analogously to specific logic circuits.

**[0018]** In order to provide context and facilitate an understanding of the invention, an overview and discussion of an illustrative event driven system will now be presented. In particular, FIG. 1 shows a block diagram of an illustrative communications network which, as will appreciated, contains a myriad of event driven applications. Communications network 100, e.g., is a public switched telephone network such as the well-known inter-exchange network of AT&T Corp., that provides long distance telephone services for its subscribers. These subscribers access communications network 100 through, e.g., communications devices 105-1 through 105-6, which are, e.g., customer premise equipment, wired telephones, personal computers, cellular telephones, pagers and facsimile machines. Communications network 100 includes, *inter alia*, a plurality of toll switches, e.g., shown in FIG. 1 as toll switches 115, 120, 125, and 130. These toll switches may be any of the well-known types of telecommunications switching equipment, e.g., the No. 4ESS® (Electronic Switching System) or the No. 5ESS® available from Lucent Technologies Inc., 600 Mountain Ave., Murray Hill, NJ 07974. As shown in FIG. 1, each of toll switches 115, 120, 125, and 130 are connected to a number of other switches via a so-called inter-toll network shown as block 150. Each toll switch may also be connected to multiple central offices ("CO"), e.g., CO's 110 through 114. The operation of such telecommunications networks and CO's is well-known, e.g., as discussed in "Engineering and Operations in the Bell System", Second Edition, Eighth Printing, International Standard Book Number 0-932764-04-5, 1993, and the detail of which will not be further discussed herein. In short, a CO is arranged to extend to a corresponding toll switch of communications network 100 a telephone call originating at, e.g., communications device 105-1, from which a calling party has dialed a particular telephone number. The CO, e.g., CO 110, is further arranged to extend the call connection to, e.g., communications device 105-6 associated with the called party and to the CO, e.g., CO 114, which receives the extension of the call from the corresponding toll switch, e.g., toll switch 125.

**[0019]** Toll switches 115, 120, 125 and 130 of communications network 100 are interconnected via data link 135, which may be, e.g., the well-known System Signaling 7 ("SS7") network. Communications network 100 is arranged so that the toll switches may exchange data messages with one another to establish a connection between a calling party (e.g., communications device 105-1) and a called party (e.g., communications device 105-6), via communications network 100. That is, the connection is made by extending a call through communications network 100 until the call is completed (e.g., the called party answers the call by going "off-hook") between the calling party and the called party. Communications network 100 further includes a number of centralized databases commonly known as Network Control Points ("NCPs"), a single one of which is shown as NCP 140. As is well-known, NCP's, such as NCP 140, are strategically positioned at various positions within communications network 100 to support various service features accessed and provided through the network such as the well-known "800" or "888" toll-free telephone number services.

**[0020]** As will be appreciated, communications network 100 and the various above-described elements of the network require a myriad of event driven applications. For example, call processing application programs for toll switches 115, 120, 125 and 130 must be developed to facilitate the extension of a call through network 100. In particular, for example, a calling party may wish to utilize communications device 105-1 for placing an outgoing call. Such an event driven call-processing application will include operations, i.e., control states, such as offhook, dial, idle, connect, ring, and busy, to name just a few.

**[0021]** As mentioned above, in accordance with the preferred embodiment of the invention, event driven software code is specified and written in accordance with the principles of the previously cited, co-pending, commonly assigned patent application, entitled "Method and Apparatus For Developing Event Driven Software" (hereinafter referred to as "the Method and Apparatus For Developing Event Driven Software application"). Thus, in order to further facilitate a complete understanding of the present invention, the various aspects of the invention in "the Method and Apparatus For Developing Event Driven Software application" will be now be discussed in some detail.

**[0022]** The various aspects of the invention in that application, provide a programming technique for defining, structuring, and developing event driven software. More particularly, event driven software is structured in two basic parts which are referred to therein and herein specifically as control-sensitive code and non-control-sensitive code. In accordance with the invention, control sensitive code includes the identification of control states, the events that are generated and that must be recognized, and the responses to the arrival of particular events in particular states. Further, within the control-sensitive code defining the system, control states are uniquely identified throughout such program code with a single symbol. The non-control-sensitive code, specified separately from the control sensitive code, is directed at the necessary normal sequential code for programming the event driven system such as the invocation of device drivers, the control of peripheral devices, the updating of tables, variable definition, etc. Advantageously, the invention allows the software developer to structure the event driven software application code such that a clear distinction is made between core control features, i.e., states, events and responses, and extraneous details not directly

impacting the behavior of the event driven system. As such, the programmer can determine the exact execution flow from control state-to-control state directly from a cursory examination of the program code itself as the individual control states have a strong visual context. That is, in accordance with the invention, the program source code is structured and defined such that the control states which define the event driven application have a direct and logical, i.e. causal, relationship which is immediately evident from an examination of the source code itself.

5 [0023] More particularly, FIG. 2 and FIG. 3 show an example of a conventional C source code program 200 (delimited by individual program code line numbers 1-268) directed to such an illustrative event driven application for processing an outgoing call. As will be appreciated, source code program 200 defines several control states for the application. See, for example, code sections 205 through 230 which define operations for the following respective 10 control states: busy, conn, dead, digits, idle, and ring. Significantly, however, an examination of source code program 200 does not provide an immediately evident view of the execution thread between the various control states. From a high level programming perspective, there is no direct or logical, i.e. causal, relationship between the various control states defined within the source program code. That is, as mentioned above, in order for a programmer to understand the logical flow from control state-to-control state directly from the program source code it is necessary to trace the 15 execution thread through the source, e.g., tracing branch instructions or "goto" statements. As will be appreciated, the tracing of such logical flow, particularly in very large source code programs, is a tedious, unilluminating, error prone, and time consuming task. For example, the source code for implementing all the requisite call processing features of communications network 100 will most likely be on the order of tens of thousands lines of code. This also leads to certain complexity in the testing of the software for so-called "bugs" and to determine whether the application program 20 implements the desired features. Significantly, the novel programming technique is directed to defining, structuring, and coding event driven software which mitigates certain disadvantages of the prior art. In accordance with the invention, control sensitive code includes the identification of control states, the events that are generated and that must be recognized, and the responses to the arrival of particular events in particular states. Further, within the control-sensitive code defining the system, control states are uniquely identified throughout such program code with a single symbol.

25 The non-control-sensitive code, specified separately from the control sensitive code, is directed at the necessary normal sequential code for programming the event driven system such as the invocation of device drivers, the control of peripheral devices, the updating of tables, variable definition, etc.

30 [0024] Advantageously, the invention allows the software developer to structure the event driven software application code such that a clear distinction is made between core control features, i.e., states, events and responses, and extraneous details not directly impacting the behavior of the event driven system. As such, the programmer can determine the exact execution flow from control state-to-control state directly from a cursory examination of the program code itself as the individual control states have a strong visual context. That is, in accordance with the invention, the program source code is structured and defined such that the control states which define the event driven application have a direct and logical, i.e. causal, relationship which is immediately evident from an examination of the source code itself.

35 [0025] In accordance with the preferred embodiment of the invention, a programming extension to the well-known ANSI-standard C programming language is adopted for facilitating the definition, structure and coding of event driven software in accordance with the invention. More particularly, in accordance with the preferred embodiment of the invention, the extension to the ANSI-standard C programming language consists of a single symbol, illustratively chosen to be the well-known ASCII symbol "@", and used for labeling all control states within the control sensitive portion of 40 the program code. That is, the symbol "@" is used as the initial character of all labels that correspond to control states within the program source code. Thus, in accordance with the invention, a program statement prefixed with the "@" label identifies a control state in the program code whereby the current execution of the event driven system waits for the arrival of the next event. Following the arrival of the next event, execution of the subsequent program code follows along with the encoding of the required response to that event, and concluding with a transition to the new, uniquely 45 defined control state. As will be appreciated, the response and the new control state, can be different for each event being processed depending upon operational issues such as any state information stored in internal tables, variables, device responses, and the like.

50 [0026] FIG. 4 shows a flowchart of illustrative operations for software development in accordance with the principles of the invention. More particularly, a specification for the event driven system to be programmed is defined (block 410), e.g., by the system engineer. As will be appreciated by those skilled in the art, such a specification provides a description of the overall application, e.g., event driven system, for which a software program is to be written and implemented. From the system specification, the control states for implementing the event driven system under development can be identified (block 420) which, in turn, leads to the identification of the events which will be generated and that must be recognized, and the responses to the arrival of particular events in particular states. As such, in accordance with the 55 invention, the definition, structuring, and coding (block 430) of the source code program is divided into two parts, as discussed above, the control-sensitive code and the non-control-sensitive-code.

[0027] In accordance with the invention, within the control-sensitive code defining the system, control states are uniquely identified throughout such program code with a single symbol. As discussed above, in accordance with the

preferred embodiment of the invention, the symbol "@" is used as the initial character of all labels that correspond to control states within the control-sensitive portion of the overall program source code. Thus, in accordance with the invention, a program statement prefixed with the "@" label identifies a wait-state in the program code, where the current execution of the event driven system waits for the arrival of the next event. Further, the non-control-sensitive code,

5 specified separately from the control sensitive code, is directed at the necessary normal sequential code for programming the event driven system such as the invocation of device drivers, the control of peripheral devices, the updating of tables, variable definition, etc.

10 [0028] Advantageously, the invention allows the software developer to structure the event driven software application code such that a clear distinction is made between core control features, i.e., states, events and responses, and extraneous details not directly impacting the behavior of the event driven system. Thus, one practical result of applying the principles of the invention is the programmer can determine the exact execution flow from control state-to-control state directly from a cursory examination of the program code itself as the individual control states have a strong visual context. That is, in accordance with the invention, the program source code is structured and defined such that the control states which define the event driven application have a direct and logical, i.e. causal, relationship which is immediately evident from an examination of the source code itself. Significantly, our present invention leads to further advantages in the testing and verifying of the event driven software (block 440) to determine if the software satisfies the desired properties as specified by the user. In particular, it is the testing of such event driven software that the present invention is directed and further discussed below in greater detail.

15 [0029] However, continuing with the discussion of "the Method and Apparatus For Developing Event Driven Software application", as mentioned above, that invention allows the software developer to structure the event driven software application code such that a clear distinction is made between core control features, i.e., states, events and responses, and extraneous details not directly impacting the behavior of the event driven system. For example, FIG.'s 5 and 6 show an illustrative C source code program 500 (delineated by individual program code line numbers 1-258), structured and defined in accordance with the invention, for the illustrative telecommunications call processing application of FIG. 2 and 3. In particular, control states 510 through 555 are delineated, in accordance with the preferred embodiment, with the symbol "@" as the initial character of all labels thereby designating the control states within the control-sensitive portion of the overall source code program 500. Thus, source code program 500, in accordance with the invention, has a direct and logical, i.e. causal, relationship between the individual control states, i.e., control states 510-555. Significantly, the programmer can determine the exact execution flow from control state-to-control state directly from a cursory examination of the program code itself as the individual control states have a strong visual context.

20 [0030] For example, through a cursory examination of source code program 500, one can quickly and exactly ascertain when viewing the code at control state 535, i.e., the "@digits" control state, how that particular control state is reached. That is, invocation and execution of this particular event driven system, i.e., the processing of an outgoing call, a control state for allowing the dialing of digits on a telephone device is processed. Control state 535 is such a state and is clearly identifiable beginning at line 125 of code 500, in accordance with preferred embodiment of the invention. In viewing control state 535, one can readily see that the execution flow in reaching this control state is directly through the above-coded control states 510 through 530, respectively. Further, if source code program 500 needs to be revised to add another control state, e.g., to add some further application-specific functionality, the new control state can be placed at the exact location within the existing control state hierarchy without losing context between all the control states.

25 [0031] This logical, i.e. causal, relationship among control states and the clear execution flow directly perceived from viewing source code program 500 is further evident in FIG.'s 7 and 8 which show a translated source code program 700 derived from the illustrative C source code program 500 of FIG.'s 5 and 6. As mentioned above, in accordance with the preferred embodiment of the invention, the extension to the ANSI-standard C programming language consists of a single symbol illustratively chosen to be the symbol "@" and used for labeling all control states within the control sensitive portion of the program code. The mechanics, i.e. the translator, of the actual translation of the code itself will be readily apparent to those skilled in the art. Thus, translated source code program 700 is the actual fully translated ANSI-standard C representation of the event driven application program in accordance with the invention. For example, control state 515 (i.e., "@idle") of FIG. 5 is translated to program code section 710, and control state 545 (i.e., "@ring") is translated to program code section 720.

30 [0032] The above-described advantages of the invention in "the Method and Apparatus For Developing Event Driven Software application" provide significant utility in the computer programming arts, in particular, to the definition, structure, and development of event driven software application programs. Of course, critical to the successful development of any application program is the effective testing and verification of the program code against a defined set of expected properties. It is the testing of event driven software that the various aspects of the present invention are directed and which now will be discussed in greater detail.

35 [0033] The various aspects of the present invention are directed to the efficient testing of event driven software and making it possible to track the validity of future changes to the program code, as part of the normal maintenance,

extension and revision process to the source code. FIG. 9 shows a flowchart of illustrative operations for testing event driven software in accordance with the principles of the invention. More particularly, in accordance with the preferred embodiment of the invention, event driven software code is specified and written in accordance with the principles of the invention in "the Method and Apparatus For Developing Event Driven Software application", as described previously.

5 That is, in accordance with the preferred embodiment, the source code program is defined and structured into control-sensitive and non-control-sensitive format (block 910.) In accordance with the invention, the event driven source code program is parsed to extract the control states defined in the source program, and to convert the source code into an intermediate state machine format (block 920.) Thereafter, in accordance with the invention, the intermediate state machine format is converted into a automata-based format for modeling checking purposes.

10 [0034] In particular, in accordance with the preferred embodiment, the intermediate state machine format is converted into an annotated automaton model in the input language of a specific model checker (block 940). In accordance with the preferred embodiment, the model checker tool is the well-known "SPIN" model checker developed by and available from the Bell Laboratories Division of Lucent Technologies Inc., and as described in more detail, e.g., in G. J. Holzmann, The Model Checker SPIN, *IEEE Trans. On Software Engineering*, Vol. 23, No. 5, pp. 279-295, May 1997, which is hereby incorporated by reference for all purposes. Importantly, in accordance with the invention, each statement from the source code program is mapped (block 930) onto an abstraction within the automaton model. The mapping, discussed below in greater detail, is facilitated by a translation map which is defined at the initial stage of the verification process. The map need not be revised thereafter unless new types of instructions are introduced, e.g., after revisions are made to repair programming faults, into the source program. The mapping, *inter alia*, dictates a fixed translation

20 of the particular statement into the target language of the model checker. Thus, instructions that appear in the map are systematically converted into the automaton model wherever such instructions appear in the source code.

25 [0035] In addition to the above-described map, in accordance with the invention, a so-called environment model is defined, illustratively by a user, which encapsulates a minimal set of assumptions that must be made about the particular operating environment in which the event driven application is executed. More particularly, as the model checker input language is provided (block 950) to the model checker, e.g., SPIN, the environmental model is applied (block 975). As a result, in accordance with the preferred embodiment of the invention, the verification of the properties of the event driven system is made subject to the environmental model during the testing and checking (block 960) of the event driven software by the model checker. Thereafter, the testing and checking results are output, in a well-known manner, by the model checker (block 970) for determining whether the event driven system conforms with the user's desired execution properties and behaviors.

30 [0036] Important to the above-described testing and checking of event driven software, in accordance with the invention, is the efficient parsing of the event driven source code program to extract the control states defined in the source program, and converting the source code into an intermediate state machine format (see, e.g., FIG. 9, block 920.) That is, to perform model checking efficiently, one must be able to identify the control states, events and actions (i.e., responses) of the event driven system. Thus, a further aspect of the invention is directed to our realization of a formal specification which facilitates the description of an event driven system for model checking purposes, as will now be discussed.

35 [0037] In accordance with this aspect of the invention, a format is defined which facilitates the specification of an event driven system. In accordance with various embodiments of the invention (discussed further below), the specification may be converted into a target programming language, e.g. C, for direct compilation and execution. In accordance with still further embodiments of the invention, the specification is converted, using the above-described mapping, into a logical verification model such as the input language of the SPIN model checking tool. As will be appreciated by those skilled in the art, grammar specification tools are well-known. For example, the YACC (yet-another-compiler-compiler) is one well-known UNIX software utility tool (see, e.g., A. T. Schreiner, *Using C with curses, lex and yacc*, 40 Prentice-Hall Inc., Englewood Cliffs, NJ 07632, 1990) which assists programmers in the development of C routines which analyze and interpret an input stream, and also facilitates the development of compilers and interpreters. Similar to YACC, our specification format provides a formalism for describing the grammar of a language, and automating the generation of a compiler for that language. However, our specification format, in accordance with the invention, is directed towards the specification of the behavior of the system.

45 [0038] More particularly, an illustrative grammar definition, in accordance with the preferred embodiment of the invention is as follows:

```
5      spec:  defs “%%” rules
         ;
10     defs:  /*empty*/
         |      one_def
         |      one_def defs
         ;
15     one_def:  “%c_template” “\” “ STRING ”\”
         |      “%p_template” “\” “ STRING ”\”
         |      “%p_map” “\” “ STRING ”\”
         |      “%nparam”NUMBER
         |      “%event” names
         |      “%state” names
         ;
30     names:  STRING
         |      STRING names
         ;
35
rules: one_rule
         |      one_rule rules
40         ;
         ;
45     one_rule:  source “:” opt_recpt responses ”;”
         ;
         ;
50     responses:  one_response
         |      one_response “|” responses
one_response:      triggers opt_target opt_action
55         ;
```

```

triggers:      one_trigger
5             | one_trigger "+" triggers
;
;
one_trigger:event
10            | "("condition")"
| "!("condition")"
| "else"
15            | "always"
;
;
20            opt_target: /*empty*/
| target
;
;
25            opt_action: /*empty*/
| "{"action"}"
;
;
30            opt_recpt: /*empty*/
| onreceipt "{"action"}"
;
;
35            event:      STRING;
40            condition:  STRING;
            action:      STRING;
            source:      STRING;
45            target:      STRING | "@" STRING;

```

---



---

50 The above illustrative grammar definition provides the framework for defining a specification for a particular application, e.g., an event driven system, in accordance with the principles of the invention. All items in the illustrative grammar definition which appear in double quotes are terminal symbols (keywords, literals, tokens). The capitalized word STRING represents an alphanumeric string, and NUMBER represents a sequence of one or more digits. Alternatives are separated by a vertical bar "|", i.e., pipe symbol, and each grammar rule is terminated by a semicolon.

55 [0039] In accordance with the preferred embodiment, the specification format consists of two parts: a declaration and a specification. The declaration part of the specification of the preferred embodiment, contains, *inter alia*, a listing of symbolic names for all events that will be used in the specification. For example, the following code fragment is an

illustrative declaration in accordance with the preferred embodiment:

5

---

```

 5      %c_template      "c.ctp" /*optional: a C code template */
 10     %p_template      "p.ctp" /*optional: a SPIN model template */
 15     %p_map           "p.map" /* optional: maps C code to SPIN */
 20
 25     %event Cranswer
 30     %event Crbusy
 35     %event Crconn
 40     %event Crdigit
 45
 50     %state S_idle
 55     %state S_pots
 60     %state S_alloc
 65     %state S_parked
 70     %state S_busy
 75
 80
 85
 90
 95
 100
 105
 110
 115
 120
 125
 130
 135
 140
 145
 150
 155
 160
 165
 170
 175
 180
 185
 190
 195
 200
 205
 210
 215
 220
 225
 230
 235
 240
 245
 250
 255
 260
 265
 270
 275
 280
 285
 290
 295
 300
 305
 310
 315
 320
 325
 330
 335
 340
 345
 350
 355
 360
 365
 370
 375
 380
 385
 390
 395
 400
 405
 410
 415
 420
 425
 430
 435
 440
 445
 450
 455
 460
 465
 470
 475
 480
 485
 490
 495
 500
 505
 510
 515
 520
 525
 530
 535
 540
 545
 550
 555
 560
 565
 570
 575
 580
 585
 590
 595
 600
 605
 610
 615
 620
 625
 630
 635
 640
 645
 650
 655
 660
 665
 670
 675
 680
 685
 690
 695
 700
 705
 710
 715
 720
 725
 730
 735
 740
 745
 750
 755
 760
 765
 770
 775
 780
 785
 790
 795
 800
 805
 810
 815
 820
 825
 830
 835
 840
 845
 850
 855
 860
 865
 870
 875
 880
 885
 890
 895
 900
 905
 910
 915
 920
 925
 930
 935
 940
 945
 950
 955
 960
 965
 970
 975
 980
 985
 990
 995
 1000
 1005
 1010
 1015
 1020
 1025
 1030
 1035
 1040
 1045
 1050
 1055
 1060
 1065
 1070
 1075
 1080
 1085
 1090
 1095
 1100
 1105
 1110
 1115
 1120
 1125
 1130
 1135
 1140
 1145
 1150
 1155
 1160
 1165
 1170
 1175
 1180
 1185
 1190
 1195
 1200
 1205
 1210
 1215
 1220
 1225
 1230
 1235
 1240
 1245
 1250
 1255
 1260
 1265
 1270
 1275
 1280
 1285
 1290
 1295
 1300
 1305
 1310
 1315
 1320
 1325
 1330
 1335
 1340
 1345
 1350
 1355
 1360
 1365
 1370
 1375
 1380
 1385
 1390
 1395
 1400
 1405
 1410
 1415
 1420
 1425
 1430
 1435
 1440
 1445
 1450
 1455
 1460
 1465
 1470
 1475
 1480
 1485
 1490
 1495
 1500
 1505
 1510
 1515
 1520
 1525
 1530
 1535
 1540
 1545
 1550
 1555
 1560
 1565
 1570
 1575
 1580
 1585
 1590
 1595
 1600
 1605
 1610
 1615
 1620
 1625
 1630
 1635
 1640
 1645
 1650
 1655
 1660
 1665
 1670
 1675
 1680
 1685
 1690
 1695
 1700
 1705
 1710
 1715
 1720
 1725
 1730
 1735
 1740
 1745
 1750
 1755
 1760
 1765
 1770
 1775
 1780
 1785
 1790
 1795
 1800
 1805
 1810
 1815
 1820
 1825
 1830
 1835
 1840
 1845
 1850
 1855
 1860
 1865
 1870
 1875
 1880
 1885
 1890
 1895
 1900
 1905
 1910
 1915
 1920
 1925
 1930
 1935
 1940
 1945
 1950
 1955
 1960
 1965
 1970
 1975
 1980
 1985
 1990
 1995
 2000
 2005
 2010
 2015
 2020
 2025
 2030
 2035
 2040
 2045
 2050
 2055
 2060
 2065
 2070
 2075
 2080
 2085
 2090
 2095
 2100
 2105
 2110
 2115
 2120
 2125
 2130
 2135
 2140
 2145
 2150
 2155
 2160
 2165
 2170
 2175
 2180
 2185
 2190
 2195
 2200
 2205
 2210
 2215
 2220
 2225
 2230
 2235
 2240
 2245
 2250
 2255
 2260
 2265
 2270
 2275
 2280
 2285
 2290
 2295
 2300
 2305
 2310
 2315
 2320
 2325
 2330
 2335
 2340
 2345
 2350
 2355
 2360
 2365
 2370
 2375
 2380
 2385
 2390
 2395
 2400
 2405
 2410
 2415
 2420
 2425
 2430
 2435
 2440
 2445
 2450
 2455
 2460
 2465
 2470
 2475
 2480
 2485
 2490
 2495
 2500
 2505
 2510
 2515
 2520
 2525
 2530
 2535
 2540
 2545
 2550
 2555
 2560
 2565
 2570
 2575
 2580
 2585
 2590
 2595
 2600
 2605
 2610
 2615
 2620
 2625
 2630
 2635
 2640
 2645
 2650
 2655
 2660
 2665
 2670
 2675
 2680
 2685
 2690
 2695
 2700
 2705
 2710
 2715
 2720
 2725
 2730
 2735
 2740
 2745
 2750
 2755
 2760
 2765
 2770
 2775
 2780
 2785
 2790
 2795
 2800
 2805
 2810
 2815
 2820
 2825
 2830
 2835
 2840
 2845
 2850
 2855
 2860
 2865
 2870
 2875
 2880
 2885
 2890
 2895
 2900
 2905
 2910
 2915
 2920
 2925
 2930
 2935
 2940
 2945
 2950
 2955
 2960
 2965
 2970
 2975
 2980
 2985
 2990
 2995
 3000
 3005
 3010
 3015
 3020
 3025
 3030
 3035
 3040
 3045
 3050
 3055
 3060
 3065
 3070
 3075
 3080
 3085
 3090
 3095
 3100
 3105
 3110
 3115
 3120
 3125
 3130
 3135
 3140
 3145
 3150
 3155
 3160
 3165
 3170
 3175
 3180
 3185
 3190
 3195
 3200
 3205
 3210
 3215
 3220
 3225
 3230
 3235
 3240
 3245
 3250
 3255
 3260
 3265
 3270
 3275
 3280
 3285
 3290
 3295
 3300
 3305
 3310
 3315
 3320
 3325
 3330
 3335
 3340
 3345
 3350
 3355
 3360
 3365
 3370
 3375
 3380
 3385
 3390
 3395
 3400
 3405
 3410
 3415
 3420
 3425
 3430
 3435
 3440
 3445
 3450
 3455
 3460
 3465
 3470
 3475
 3480
 3485
 3490
 3495
 3500
 3505
 3510
 3515
 3520
 3525
 3530
 3535
 3540
 3545
 3550
 3555
 3560
 3565
 3570
 3575
 3580
 3585
 3590
 3595
 3600
 3605
 3610
 3615
 3620
 3625
 3630
 3635
 3640
 3645
 3650
 3655
 3660
 3665
 3670
 3675
 3680
 3685
 3690
 3695
 3700
 3705
 3710
 3715
 3720
 3725
 3730
 3735
 3740
 3745
 3750
 3755
 3760
 3765
 3770
 3775
 3780
 3785
 3790
 3795
 3800
 3805
 3810
 3815
 3820
 3825
 3830
 3835
 3840
 3845
 3850
 3855
 3860
 3865
 3870
 3875
 3880
 3885
 3890
 3895
 3900
 3905
 3910
 3915
 3920
 3925
 3930
 3935
 3940
 3945
 3950
 3955
 3960
 3965
 3970
 3975
 3980
 3985
 3990
 3995
 4000
 4005
 4010
 4015
 4020
 4025
 4030
 4035
 4040
 4045
 4050
 4055
 4060
 4065
 4070
 4075
 4080
 4085
 4090
 4095
 4100
 4105
 4110
 4115
 4120
 4125
 4130
 4135
 4140
 4145
 4150
 4155
 4160
 4165
 4170
 4175
 4180
 4185
 4190
 4195
 4200
 4205
 4210
 4215
 4220
 4225
 4230
 4235
 4240
 4245
 4250
 4255
 4260
 4265
 4270
 4275
 4280
 4285
 4290
 4295
 4300
 4305
 4310
 4315
 4320
 4325
 4330
 4335
 4340
 4345
 4350
 4355
 4360
 4365
 4370
 4375
 4380
 4385
 4390
 4395
 4400
 4405
 4410
 4415
 4420
 4425
 4430
 4435
 4440
 4445
 4450
 4455
 4460
 4465
 4470
 4475
 4480
 4485
 4490
 4495
 4500
 4505
 4510
 4515
 4520
 4525
 4530
 4535
 4540
 4545
 4550
 4555
 4560
 4565
 4570
 4575
 4580
 4585
 4590
 4595
 4600
 4605
 4610
 4615
 4620
 4625
 4630
 4635
 4640
 4645
 4650
 4655
 4660
 4665
 4670
 4675
 4680
 4685
 4690
 4695
 4700
 4705
 4710
 4715
 4720
 4725
 4730
 4735
 4740
 4745
 4750
 4755
 4760
 4765
 4770
 4775
 4780
 4785
 4790
 4795
 4800
 4805
 4810
 4815
 4820
 4825
 4830
 4835
 4840
 4845
 4850
 4855
 4860
 4865
 4870
 4875
 4880
 4885
 4890
 4895
 4900
 4905
 4910
 4915
 4920
 4925
 4930
 4935
 4940
 4945
 4950
 4955
 4960
 4965
 4970
 4975
 4980
 4985
 4990
 4995
 5000
 5005
 5010
 5015
 5020
 5025
 5030
 5035
 5040
 5045
 5050
 5055
 5060
 5065
 5070
 5075
 5080
 5085
 5090
 5095
 5100
 5105
 5110
 5115
 5120
 5125
 5130
 5135
 5140
 5145
 5150
 5155
 5160
 5165
 5170
 5175
 5180
 5185
 5190
 5195
 5200
 5205
 5210
 5215
 5220
 5225
 5230
 5235
 5240
 5245
 5250
 5255
 5260
 5265
 5270
 5275
 5280
 5285
 5290
 5295
 5300
 5305
 5310
 5315
 5320
 5325
 5330
 5335
 5340
 5345
 5350
 5355
 5360
 5365
 5370
 5375
 5380
 5385
 5390
 5395
 5400
 5405
 5410
 5415
 5420
 5425
 5430
 5435
 5440
 5445
 5450
 5455
 5460
 5465
 5470
 5475
 5480
 5485
 5490
 5495
 5500
 5505
 5510
 5515
 5520
 5525
 5530
 5535
 5540
 5545
 5550
 5555
 5560
 5565
 5570
 5575
 5580
 5585
 5590
 5595
 5600
 5605
 5610
 5615
 5620
 5625
 5630
 5635
 5640
 5645
 5650
 5655
 5660
 5665
 5670
 5675
 5680
 5685
 5690
 5695
 5700
 5705
 5710
 5715
 5720
 5725
 5730
 5735
 5740
 5745
 5750
 5755
 5760
 5765
 5770
 5775
 5780
 5785
 5790
 5795
 5800
 5805
 5810
 5815
 5820
 5825
 5830
 5835
 5840
 5845
 5850
 5855
 5860
 5865
 5870
 5875
 5880
 5885
 5890
 5895
 5900
 5905
 5910
 5915
 5920
 5925
 5930
 5935
 5940
 5945
 5950
 5955
 5960
 5965
 5970
 5975
 5980
 5985
 5990
 5995
 6000
 6005
 6010
 6015
 6020
 6025
 6030
 6035
 6040
 6045
 6050
 6055
 6060
 6065
 6070
 6075
 6080
 6085
 6090
 6095
 6100
 6105
 6110
 6115
 6120
 6125
 6130
 6135
 6140
 6145
 6150
 6155
 6160
 6165
 6170
 6175
 6180
 6185
 6190
 6195
 6200
 6205
 6210
 6215
 6220
 6225
 6230
 6235
 6240
 6245
 6250
 6255
 6260
 6265
 6270
 6275
 6280
 6285
 6290
 6295
 6300
 6305
 6310
 6315
 6320
 6325
 6330
 6335
 6340
 6345
 6350
 6355
 6360
 6365
 6370
 6375
 6380
 6385
 6390
 6395
 6400
 6405
 6410
 6415
 6420
 6425
 6430
 6435
 6440
 6445
 6450
 6455
 6460
 6465
 6470
 6475
 6480
 6485
 6490
 6495
 6500
 6505
 6510
 6515
 6520
 6525
 6530
 6535
 6540
 6545
 6550
 6555
 6560
 6565
 6570
 6575
 6580
 6585
 6590
 6595
 6600
 6605
 6610
 6615
 6620
 6625
 6630
 6635
 6640
 6645
 6650
 6655
 6660
 6665
 6670
 6675
 6680
 6685
 6690
 6695
 6700
 6705
 6710
 6715
 6720
 6725
 6730
 6735
 6740
 6745
 6750
 6755
 6760
 6765
 6770
 6775
 6780
 6785
 6790
 6795
 6800
 6805
 6810
 6815
 6820
 6825
 6830
 6835
 6840
 6845
 6850
 6855
 6860
 6865
 6870
 6875
 6880
 6885
 6890
 6895
 6900
 6905
 6910
 6915
 6920
 6925
 6930
 6935
 6940
 6945
 6950
 6955
 6960
 6965
 6970
 6975
 6980
 6985
 6990
 6995
 7000
 7005
 7010
 7015
 7020
 7025
 7030
 7035
 7040
 7045
 7050
 7055
 7060
 7065
 7070
 7075
 7080
 7085
 7090
 7095
 7100
 7105
 7110
 7115
 7120
 7125
 7130
 7135
 7140
 7145
 7150
 7155
 7160
 7165
 7170
 7175
 7180
 7185
 7190
 7195
 7200
 7205
 7210
 7215
 7220
 7225
 7230
 7235
 7240
 7245
 7250
 7255
 7260
 7265
 7270
 7275
 7280
 7285
 7290
 7295
 7300
 7305
 7310
 7315
 7320
 7325
 7330
 7335
 7340
 7345
 7350
 7355
 7360
 7365
 7370
 7375
 7380
 7385
 7390
 7395
 7400
 7405
 7410
 7415
 7420
 7425
 7430
 7435
 7440
 7445
 7450
 7455
 7460
 7465
 7470
 7475
 7480
 7485
 7490
 7495
 7500
 7505
 7510
 7515
 7520
 7525
 7530
 7535
 7540
 7545
 7550
 7555
 7560
 7565
 7570
 7575
 7580
 7585
 7590
 7595
 7600
 7605
 7610
 7615
 7620
 7625
 7630
 7635
 7640
 7645
 7650
 7655
 7660
 7665
 7670
 7675
 7680
 7685
 7690
 7695
 7700
 7705
 7710
 7715
 7720
 7725
 7730
 7735
 7740
 7745
 7750
 7755
 7760
 7765
 7770
 7775
 7780
 7785
 7790
 7795
 7800
 7805
 7810
 7815
 7820
 7825
 7830
 7835
 7840
 7845
 7850
 7855
 7860
 7865
 7870
 7875
 7880
 7885
 7890
 7895
 7900
 7905
 7910
 7915
 7920
 7925
 7930
 7935
 7940
 7945
 7950
 7955
 7960
 7965
 7970
 7975
 7980
 7985
 7990
 7995
 8000
 8005
 8010
 8015
 8020
 8025
 8030
 8035
 8040
 8045
 8050
 8055
 8060
 8065
 8070
 8075
 8080
 8085
 8090
 8095
 8100
 8105
 8110
 8115
 8120
 8125
 8130
 8135
 8140
 8145
 8150
 8155
 8160
 8165
 8170
 8175
 8180
 8185
 8190
 8195
 8200
 8205
 8210
 8215
 8220
 8225
 8230
 8235
 8240
 8245
 8250
 8255
 8260
 8265
 8270
 8275
 8280
 8285
 8290
 8295
 8300
 8305
 8310
 8315
 8320
 8325
 8330
 8335
 8340
 8345
 8350
 8355
 8360
 8365
 8370
 8375
 8380
 8385
 8390
 8395
 8400
 8405
 8410
 8415
 8420
 8425
 8430
 8435
 8440
 8445
 8450
 8455
 8460
 8465
 8470
 8475
 8480
 8485
 8490
 8495
 8500
 8505
 8510
 8515
 8520
 8525
 8530
 8535
 8540
 8545
 8550
 8555
 8560
 8565
 8570
 8575
 8580
 8585
 8590
 8595
 8600
 8605
 8610
 8615
 8620
 8625
 8630
 8635
 8640
 8645
 8650
 8655
 8660
 8665
 8670
 8675
 8680
 8685
 8690
 8695
 8700
 8705
 8710
 8715
 8720
 8725
 8730
 8735
 8740
 8745
 8750
 8755
 8760
 8765
 8770
 8775
 8780
 8785
 8790
 8795
 8800
 8805
 8810
 8815
 8820
 8825
 8830
 8835
 8840
 8845
 8850
 8855
 8860
 8865
 8870
 8875
 8880
 8885
 8890
 8895
 8900
 8905
 8910
 8915
 8920
 8925
 8930
 8935
 8940
 8945
 8950
 8955
 8960
 8965
 8970
 8975
 8980
 8985
 8990
 8995
 9000
 9005
 9010
 9015
 9020
 9025
 9030
 9035
 9040
 9045
 9050
 9055
 9060
 9065
 9070
 9075
 9080
 9085
 9090
 9095
 9100
 9105
 9110
 9115
 9120
 9125
 9130
 9135
 9140
 9145
 9150
 9155
 9160
 9165
 9170
 9175
 9180
 9185
 9190
 9195
 9200
 9205
 9210
 9215
 9220
 9225
 9230
 9235
 9240
 9245
 9250
 9255
 9260
 9265
 9270
 9275
 9280
 9285
 9290
 9295
 9300
 9305
 9310
 9315
 9320
 9325
 9330
 9335
 9340
 9345
 9350
 9355
 9360
 9365
 9370
 9375
 9380
 9385
 9390
 9395
 9400
 9405
 9410
 9415
 9420
 9425
 9430
 9435
 9440
 9445
 9450
 9455
 9460
 9465
 9470
 9
```

For example, the following code fragment is an illustrative control state specification in accordance with the preferred embodiment:

5

---



---

```

S_idle : Cronhook S_idle
10      {x→cwsit &= Swhoock;}

| Crflash
15      {x→cwsit |= Swhoock;}

| Croffhook @S_idle_E3
20      {x→cwsit |= Swhoock;}

| Iring @S_idle_E5
25      ;

```

---

25

In the above illustrative control state specification, the transition rule begins with the name of the state, e.g., "S\_idle", followed by a colon. After the colon, a series of possible transitions out of this state are listed. The individual transitions are separated from each other by a vertical bar "|". Each complete transition has three parts: the name of an event that triggers the response, the name of a new state that is reached after the response has been performed, i.e., "the next-state", and the response itself. The next-state part of the specification defines where control moves after the response has been completely processed, and contains the name of either a predefined control state or a so-called internal state. If two events have the same next-state and response, in accordance with further embodiments of the invention, an abbreviated notation can be used such as:

40

---



---

```

| Crflash
+ Crdis
45      + Croffhook S_idle
                  {x→cwsit |= Swhoock;}
```

---

50

Using the above noted abbreviated notation, the events Crflash, Crdis, and Croffhook will all generate the same response, i.e., the execution of the specified code fragment and a transition to the state "S\_idle".

55 [0041] As will be appreciated, the event driven system is meant to stop execution at each control state and wait for the arrival of the next event. In contrast, the event driven system passes through internal states without waiting. In addition, state names in the next-state segment of a transition rule can be prefixed with the "@" symbol (it should be noted that this particular notation is not the same as that described previously with regard to identifying control states

EP 1 014 265 A1

in accordance with the various aspects of the invention in "the Method and Apparatus For Developing Event Driven Software application") to indicate that processing of the next state should continue without waiting for a next event to arrive. In such a case, the last event remains available if an event is needed to determine the behavior of the system at a next state.

5 [0042] With reference to the internal states mentioned above, such events encode an internal, non-event related decision of the event driven system. In accordance with the preferred embodiment of the invention, the syntax in the specification for internal events is the same as for control states, except that instead of event names, general Boolean conditions in the target implementation language are used. For example, an illustrative internal state is defined as follows:

10

---

15 S\_idle\_E5: {x→drv→trunk} @S\_otrunk  
                  {x→orig=ipc→orig;  
                          ipc→orig→term = x;  
                          emitcdr{x,CDRtransit,0};}  
                  | ! {x→drv→trunk} @S\_idle\_E6  
                  ;  
20

---

30

The above illustrative internal state specifies that the value of a specific field (in this case hiding two levels inside a C data structure named "x") will decide whether control moves into state "S\_otrunk" or state "S\_idle\_E6". In the first case, the piece of action code defined after the state will be executed. In the latter case, no code is executed.

35 [0043] In accordance with the preferred embodiment, if the name of an event or condition is "else" this indicates that a default response will be taken when none of the explicitly listed event names or Boolean conditions apply. For example, the above illustrative internal state code fragment can also be written as follows:

40 S\_idle\_E5: {x→drv→trunk} @S\_otrunk  
                  {x→orig=ipc→orig;  
                          ipc→orig→term = x;  
                          emitcdr{x,CDRtransit,0};}  
                  | else @S\_idle\_E6  
                  ;  
45

---

50 55 In accordance with this alternative internal state code fragment, no code fragment is executed when the condition "{x→drv→trunk}" evaluates to "false" upon reaching this state. Instead, only a transition into state "S\_idle\_E6" will result.  
[0044] Further, with regard to the specification of the event driven system, it is sometimes useful to define a common action fragment that must be executed immediately following the reception of an event in a control state. By definition,

such a code fragment is to be executed before any of the transition rules are applied. In accordance with the preferred embodiment of the invention, the keyword "onreceipt" is employed to define such common action fragments. An illustrative example of this keyword's use is as follows:

5

10

```
S_orig : onreceipt
  {x→cwsswitch&= Swconn;
   y=callroute(x→endpt→ctxt, x→lidb→number,&i);}
  | (y == nil) @S_orig_E54
  | else @S_orig_E55
  ;
```

20

25 In the above illustrative code fragment, the state is an internal state so it can always be referred to in a next-state segment with the "@" prefix. The "onreceipt" code is immediately executed when reaching this state on a transition, and the condition "(y == nil)" is evaluated. If the condition is "true", the system moves into another internal state named "S\_orig\_E54", and if "false", the system move into internal state "S\_orig\_E55" where processing continues without waiting for a new event.

30 [0045] Finally, with regard to the specification, it is sometimes useful to bypass condition checks altogether and specify an intermediate transition state that will lead unconditionally to a successor state. In accordance with the preferred embodiment of the invention, a condition that is always true is represented by the keyword "always" as is illustrated in the following example code fragment:

35

```
40 S_enabdial : onreceipt
  {x→ndigits=0;
   x→drv→dtmf(x,1);
   x→cwswit |= Swconn;}
  | always S_dial
  ;
```

55

[0046] Our realization of the above-described formal specification in describing event driven systems for model checking purposes advantageously provides a specification format for describing the behavior of a event driven system.

[0047] To further illustrate the various aspects and advantages of the invention in testing event driven software, FIG. 10 shows intermediate state machine code 1000 (delineated by individual program code line numbers 1-124) derived, in accordance with the principles of the invention as discussed above, from the source code listing of the illustrative C source code program 500 of FIG.'s 5 and 6. As discussed above, to effectively perform model checking on event driven code, an accurate assessment and identification of the control states, events, and actions must occur. Advantageously, intermediate state machine code 1000, in accordance with the principles of the invention, provides a clear delineation amongst control states, events and actions. More particularly, in accordance with the preferred embodiment, a translator is applied to C source code program 500 to directly generate intermediate state machine code 1000. The actual construction of the translator will be readily understood by those skilled in the computer programming art and need not be discussed further herein. However, by structuring C source code program 500 in accordance with the present invention, the automatic identification of the relevant control states, events, and actions is made possible. Thus, the advantages of the present invention as set forth herein are realized.

[0048] More particularly, a careful examination of intermediate state machine code 1000 show a clear delineation of events (see, e.g., code section 1010) and control states (see, e.g., code section 1020) of the declaration part of code 1000. Further, the clear interaction amongst and between such events, control states and the related actions is shown, in accordance with the invention, by the various state specifications (see, e.g., state specifications 1030 through 1050) within intermediate state machine code 1000. As discussed above with reference to the operations of FIG. 9, in accordance with the invention, intermediate state machine code 1000 is converted to the model checker input language for executing the testing of the subject program code. This conversion is facilitated by a translation map which is defined at the initial stage of the verification process. The mapping, *inter alia*, dictates a fixed translation of the particular statement into the target language of the model checker. Thus, instructions that appear in the map are systematically converted into the automaton model wherever such instructions appear in the original source code.

[0049] FIG. 11 shows an illustrative map 1100 for the event driven system defined by the illustrative C source code program 500. Map 1100 consists of program code 1110 (delineated by individual program code line numbers 1-47) which defines how the fixed translation of particular statements in the original source code is made into the target language of the model checker. In particular, program code 1110 includes event definitions 1120 and code definitions 1130. For example, line 20 of program code 1110 maps every occurrence of the statement in the left hand column onto the abstraction given in the right hand column. In accordance with the invention, the application of map 1100 results in the derivation of model checker input language 1200 (delineated by individual program code line numbers 1-303) shown in FIG.'s 12 and 13. Further, in accordance with the invention, a set of library functions is generated, illustratively shown as library functions 1400 in FIG. 14, for the model checker input language of FIG.'s 12 and 13. As will be appreciated, library functions 1400 (delineated by individual program code line numbers 1-153) provide the necessary semantics for defining terms in model checker input language 1200. In accordance with the preferred embodiment, model checker input language 1200 is directed to and useful with the SPIN model checker tool.

[0050] Significantly, one important feature of using maps, in accordance with the invention, is the capability provided to the user in defining precisely the appropriate level of detail at which the checking process will be applied, in a source independent manner. The map can contain "true" or "skip" as the translation (i.e., right hand column of program code 1110 described above) of any statement which indicates that this detail is abstracted away. Therefore, the map acts as a "filter" to filter out extraneous detail, in addition to, acting as a converter from the target programming language, e.g., C, to the verification model. The map also serves as a complete formalization of the linkage between source code and the verification model.

[0051] As described above, in accordance with the preferred embodiment of the invention, the verification of the properties of the event driven system is made subject to an environmental model during the testing and checking (see, FIG. 9, block 975) of the event driven software by the verification system. The environmental model formalizes particular assumptions, made by the user, about the environment in which the event driven program code is to be executed. For example, in the case of the illustrative telecommunications call processing application described herein, the environmental model might include formalizations directed to subscriber behavior, hardware responses, and communications network performance. In particular, FIG. 15 shows an illustrative environmental model 1500 (delineated by individual program code line numbers 1-80) for the telecommunications processing application described in FIG.'s 5 and 6. For example, code fragment 1510 of environmental model 1500 describes a formalization, i.e. a template, for a telecommunications device useful in the illustrative telecommunications application. In accordance with the preferred embodiment, the model checker, i.e., SPIN, will use model checker input language 1200 in conjunction with environmental model 1500 to determine whether the event driven system conforms with the user's desired execution properties and behaviors.

[0052] Advantageously, the separate definition of the map and environmental model, in accordance with the invention, ensure that little, if anything, in the verification infrastructure needs to be updated when routine changes are made in the original source code. That is, unless new basic types of statements are introduced in the source code, the verification process can be repeated without user intervention and the source code may be checked for continued

compliance against a large library of essential correctness properties. Further, in accordance with the invention, a precise automaton model representative of the event driven code under test is automatically generated directly from the source code and verified with a logic model checker, e.g., SPIN.

5 [0053] For example, FIG. 16 shows an illustrative automaton model 1600 for a particular control state defined by the telecommunications processing application described in FIG.'s 5 and 6. More particularly, the control state depicted in FIG. 16 is the so-called "ring" control state defined in source code program 500 as control state 545 (see, FIG.'s 5 and 6). The "@ring" program statement in line 183 of source code program 500 begins the clear delineation of this particular control state from a source program viewpoint. In accordance with the testing and checking aspects of the invention, as described above, from this control state a precise automaton, e.g., automaton model 1600, is extracted. Automaton model 1600 clearly shows the transitions in and out of the "ring" control state 1610 and amongst other control states of the application, i.e., "digits\_1" control state 1620, "idle" control state 1630, "conn" control state 1640, and "error" control state 1650. Automaton model 1600 also clearly illustrates the various events, i.e., event 1660 through event 1695, occurring in the illustrative application. Further, full automaton model 1700, shown in FIG. 17, is an illustrative full automaton model of the complete telecommunications processing application described in FIG.'s 5 and 6, and automatically generated in accordance with the invention.

10 [0054] The above-described embodiments of the invention employ a C source program written (see, e.g., FIG.'s 5 and 6) in accordance with the principles of the "the Method and Apparatus For Developing Event Driven Software application", and tested in accordance with the principles of the present invention as described above. We have further realized that certain types of users, e.g. verification system programmers or software testing engineers, may wish to 15 begin their system design process by defining the event driven in accordance with the specification format described above. For example, such users may employ the specification format set forth herein to directly code the subject event driven system. Thus, in accordance with further embodiments of the invention, the specification is converted into a target programming language, e.g. C., for direct compilation and execution. In accordance with such further embodiments of the invention, a template is employed to facilitate such a conversion. The following is an illustrative C code 20 template for use in converting this such state machine code to standard C for compilation and execution:

---



---

30

```

void
state(Line *x, Ipc *ipc)
35 {     Line *y;
        int op, i, c, os, time;

40     goto start;

```

45

50

55

```

5      /* *** template: ***/
6      @C
7
8
9
10     out:
11
12     if (x == nil)
13
14     return;
15
16     if (time != 0)
17
18     {
19
20         x->time=rtms() + time;
21
22         addtimer(x);
23
24     }
25
26     return; /* exit-point */
27
28     start:
29
30         op=ipc->type; os=x->state; time=0; switch (os)
31
32         {
33
34             @@
35
36         }
37
38         os=0;
39
40         error:
41
42             y=x->term; x->term=nil; goto Aerror;
43
44     }

```

---



---

45 [0055] The above-described C code template provides the framework for a complete conversion of the state machine specification to C. As will be appreciated, the illustrative C code template can be executed by the model checker tool itself or on a separate machine. The template includes type definitions for the arguments that are passed to a routine called "state" that is used as the core of the conversion implementation. The routine must be called whenever an event occurs that relates to the subject event driven system. The template code further contains a place-holder "@C" for the state machine that is generated from the transition rules specified in accordance with the invention. The actual program code necessary for the generation of the state machine will be readily understood by those skilled in the art and need not be further detailed herein. At each call, the "start" routine executes and retrieves an event type and parameter value from the data structure that is used by the environment to store detailed information about the current event. The current state of the process is retrieved from the data structure in the first argument to the routine, and a so-called "C switch" is performed on this state value to establish the proper position in the transition rules. The short-hand symbol "@ @" of the template is expanded by the tool into a full list of names of all control states, with matching jumps to the places in the generated, i.e., converted, C code where the code fragments for the transition rules of each state are

placed. Advantageously, in accordance with these further embodiments of the invention, the specification is converted into a target programming language, e.g., C, for direct compilation and execution.

**[0056]** For example, FIG. 18 shows an illustrative computer system 1800 useful in executing and testing event driven software code in accordance with the present invention. In particular, computer system 1800 includes conventional personal computer 1810 which is connected to display 1820 and keyboard 1830. As will be appreciated, information produced by personal computer 1810, e.g., during execution of particular event driven software code, may be displayed on display 1820 for access by a user of computer system 1800. The user, as a result of viewing display 1820, interacts with and controls personal computer 1810, in a conventional manner, using keyboard 1830. Further, mouse 1880, or other well-known pointing devices, may allow the user to provide additional inputs to personal computer 1810. Illustrative computer system 1800 further includes processor 1840 which, illustratively, executes and tests event driven application software code in accordance with the present invention. For example, such event driven application software code may be loaded into personal computer 1810 via conventional diskette 1870 and thereafter stored in internal disk memory 1860. Further, execution and testing of the code may require access and use of random access memory 1850 ("RAM") in a conventional manner. Thus, the user of computer system 1800 can execute and test software, in accordance with the present invention, to provide a wide variety of applications, e.g., an event driven system. Further, as will be appreciated, computer system 1800 may also be connected, in a conventional manner, to other computer systems or devices for execution and testing of the particular application programs.

**[0057]** For example, computer system 1800 can be used to execute and test software, in accordance with the present invention, resulting in error trace 1900 as shown in FIG. 19. Error trace 1900 consists of program code 1910 (delineated by individual program code line numbers 1-47) and is an illustrative error trace, generated in accordance with the principles of the invention, for the telecommunications processing application described in FIG.'s 5 and 6. More particularly, the illustrative error trace 1900 shows how the checking process produces a scenario which directly identifies the presence of errors in the original source code of the event driven system. Further, error trace 1900 through program code 1910 identifies such errors in terms of programming level statements, e.g., C-level statements, that may be executed to reproduce the error. In the illustrative error trace 1900, the trace demonstrates in a direct sequence of C-program code statement (see, FIG. 9, program code 1920) executions that leads the event driven system into the illustrate state "@busy" with "Crdigit" being the first event to be processed. As can be seen and appreciated from error trace 1900, in conjunction with, C source program code 500 (in particular, control state 555 at line 223), there is no provision in the program code, i.e., a coding error, to anticipate the possible occurrence of this event. Advantageously, in accordance with the invention, a user can interpret error trace 1900 solely in terms of C-level execution statements without any specialized knowledge of the model itself or significant parts of the intermediate representations that are used in the checking process itself.

**[0058]** The foregoing merely illustrates the principles of the present invention. It will thus be appreciated that those skilled in the art will be able to devise various arrangements which, although not explicitly described or shown herein, embody the principles of the invention and are included within its scope. Furthermore, all examples and conditional language recited herein are principally intended expressly to be only for pedagogical purposes to aid the reader in understanding the principles of the invention and the concepts contributed by the Applicant(s) to furthering the art, and are to be construed as being without limitation to such specifically recited examples and conditions. Moreover, all statements herein reciting principles, aspects, and embodiments of the invention, as well as specific examples thereof, are intended to encompass both structural and functional equivalents thereof. Additionally, it is intended that such equivalents include both currently known equivalents as well as equivalents developed in the future, i.e., any elements developed that perform the same function, regardless of structure.

**[0059]** Thus, for example, it will be appreciated by those skilled in the art that the block diagrams herein represent conceptual views of illustrative circuitry embodying the principles of the invention. Similarly, it will be appreciated that any flowcharts, flow diagrams, state transition diagrams, pseudocode, program code, and the like represent various processes which may be substantially represented in computer readable medium and so executed by a computer, machine, or processor, whether or not such computer, machine, or processor, is explicitly shown.

**[0060]** Further, in the claims hereof any element expressed as a means for performing a specified function is intended to encompass any way of performing that function, including, for example, a) a combination of circuit elements which performs that function; or b) software in any form (including, therefore, firmware, object code, microcode or the like) combined with appropriate circuitry for executing that software to perform the function.

## Claims

1. A software testing method, carried out on a source program of the software, the source program including a plurality of instructions, said method comprising:

5       parsing the source program to identify a plurality of control states and converting the source program to a state machine format;  
      applying a translation map to the plurality of instructions of the source program;  
      converting the state machine format of the source program to a verification program;  
      applying a environmental model to the verification program; and  
      testing the software in a verification tool using the verification program.

10      2. The software testing method of claim 1 wherein the source program contains a first source code segment for causing a computer to execute a event driven system; the first source code segment including a first plurality of instructions defining the plurality of control states, the plurality of control states being associated with the event driven system, a second plurality of instructions defining a plurality of events of the event driven system, and a third plurality of instructions defining a plurality of actions of the event driven system, each control state of the plurality of control states having a individual identifier within the first plurality of instructions of the first source code segment, the individual identifier being a programming symbol which is common to all the identified control states, and a second source code segment containing at least one plurality of instructions defining a sequential control of the computer executing the event driven system.

15      3. The software testing method of claim 1 or claim 2 wherein the translation map includes a plurality of mapping instructions.

20      4. The software testing method of claim 3 wherein the applying the translation map operation includes:  
                  comparing the plurality of mapping instructions to the plurality of instructions of the source program to detect a match; and  
                  25        converting particular ones of the mapping instructions which match particular ones of the instructions of the source program to the input language of the model checker.

25      5. The software testing method of any of the preceding claims wherein the environmental model includes a plurality operational attributes of the event driven system.

30      6. The software testing method of any of the preceding claims wherein the verification tool is a model checker tool.

35      7. The software testing method of claim 6 wherein the model checker tool is a SPIN model checker tool.

40      8. The software testing method of any of the preceding claims wherein the event driven system is a telecommunications service application, the telecommunications service application having at least one event associated with extending a telephone call through a communications network.

45      9. The software testing method of any of the preceding claims wherein the state machine format is converted using a specification defined by a user.

50      10. The software testing method of any of the preceding claims further comprising:  
                  compiling the first source code segment and the second source code segment into an object program; and  
                  executing the object program to provide the event driven system.

55      11. The software testing method of any of the preceding claims wherein the step of testing uses at least one set of properties defined by a user associated with the execution of the event driven system.

60      12. The software testing method of claim 11 wherein the translation map is specified by the user.

65      13. The software testing method of claim 12 wherein the translation map, the environmental model, and the series of programming statements are each written in the C programming language.

70      14. The software testing method of any of the preceding claims including the step of displaying a result of the testing of the software.

75      15. The software testing method of claim 14 wherein the displayed result includes at least one full automaton repre-

sentation of the event driven system.

16. A software testing apparatus comprising means arranged to carry out each step of a method as claimed in any of the preceding claims.

5

17. A machine-readable medium having stored thereon a plurality of instructions, the plurality of instructions including instructions that, when executed by a machine, cause the machine to perform a method as claimed in any of claims 1 to 15.

10 18. A software testing system comprising:

a computer having at least one memory;

means for receiving a source program of software to be tested and comprising series of programming language statements in the memory;

15 and means arranged to carry out each step of a method as claimed in any of claims 1 to 15 to test said software.

20

25

30

35

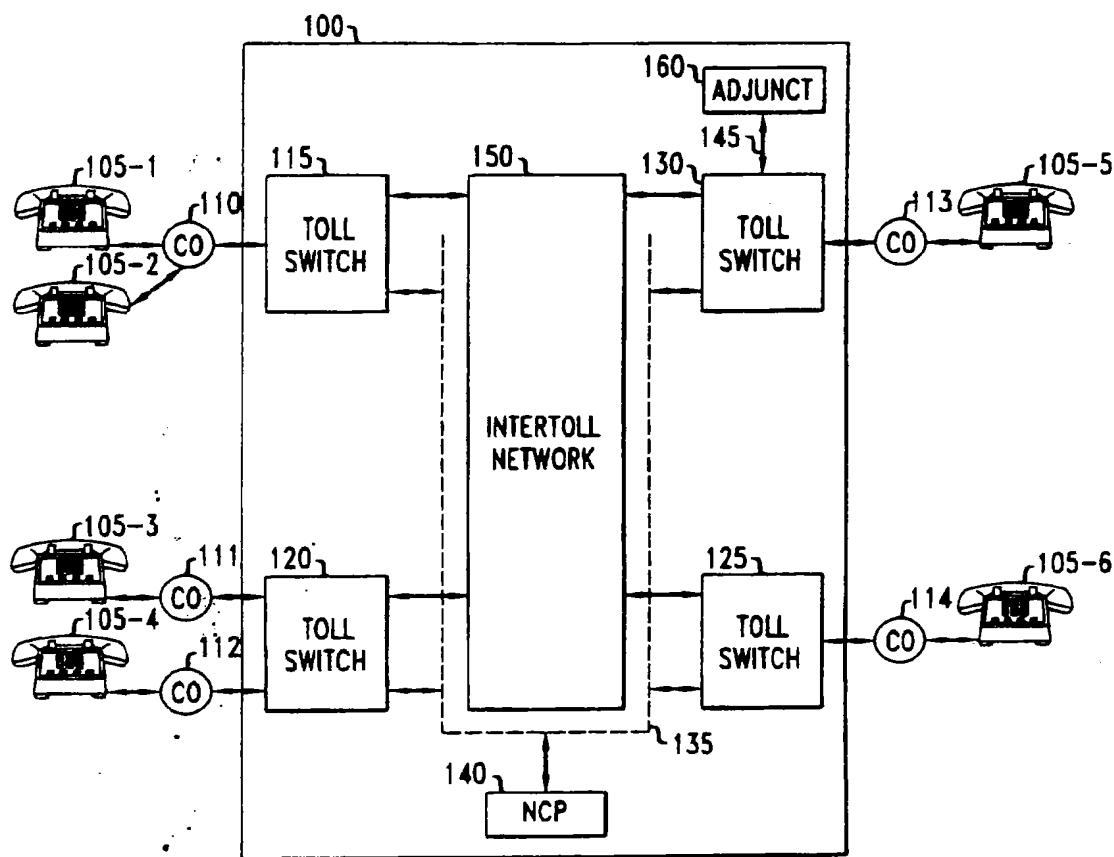
40

45

50

55

FIG. 1



200

FIG. 2

1 //equivalent of state.z in traditional notation  
 2 //first switch on current state, then on event

```

  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115

```

89 case conn: //phones are connected  
 90 switch(op) {  
 91 default: //successful connection  
 92 case Cconn: break;  
 93 // no state change  
 94 case Cronhook:  
 95 send(x->term, Cdis, 0);  
 96 send(x->term, Cdis, 0);  
 97 send(x, Cdis, 0);  
 98 x->state = idle;  
 99 break;  
100 }  
101 break;  
102 //phone unassigned  
103 case dead: //phone unassigned  
104 switch(op) {  
105 //check the event  
106 default:  
107 print("%! dead and op=%s\n", x, istring(op));  
108 break; //no state change  
109 case Crprov:  
110 line provisioned //line provisioned  
111 x->term = 0; //initialize  
112 x->state = idle; //next state  
113 break;  
114 break;  
115 break;

10 FIG 3

FROM FIG 2

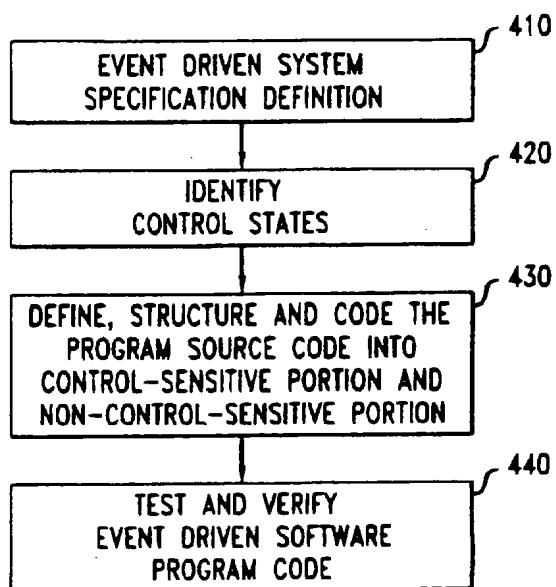
FIG. 3

```
116 case digits:
117   x->time = seconds(6);
118   switch(op) {
119     default:
120       goto error;
121     case Cronhook: //line was hung up
122       send(x, Cdis, 0);
123       x->state = idle;
124       break;
125     case Crlone: //noise (dial tone or silence has been applied)
126       break;
127     case Crdigit: //next digit detected
128       //decode digits according to dial plan
129       break;
130     case Crdtmi: //noise (dtmf has been removed)
131       //no state change
132       p = dialplan(x->digits, x->ndigits);
133       if(x->ndigits == 1) //first digit
134         send(x, Cnone, 0); //remove dial tone
135       if(p == 0) //incomplete
136         break;
137       break;
138     // translate local number to local phone
139     case ring: //one phone has audible ringing, other phone
140     if(x->term == 0) {
141       x->term = lookup(p);
142       send(x, Ctlone, reorder);
143       x->state = busy;
144     } else
145       send(x->term, Cring, 1); //ring destination
146       x->state = digits_2;
147     break;
148   }
149   case Crlone:
150   case Crlone:
151   case Crlone:
152   case Crlone:
153   case Crlone:
154   case Crlone:
155   case Crlone:
156   case Crlone:
157   case Crlone:
158   case Crlone:
159   case Crlone:
160   case Crlone:
161   case Crlone:
162   case Crlone:
163   case Crlone:
164   case Crlone:
165   case Crlone:
166   case Crlone:
167   case Crlone:
168   case Crlone:
169   case Crlone:
170   case Crlone:
171   case Crlone:
172   case Crlone:
173   case Crlone:
174   case Crlone:
175   case Crlone:
176   case Crlone:
177   case Crlone:
178   case Crlone:
179   case Crlone:
180   case Crlone:
181   case Crlone:
182   case Crlone:
183   case Crlone:
184   case Crlone:
185   case Crlone:
186   case Crlone:
187   case Crlone:
188   case Crlone:
189   case Crlone:
190   case Crlone:
```

© 1998 Lucent Technologies Inc.

→ TO FIG 5

*FIG. 4*



FROM FIG 3

L4-10 FIG 6

90	case Craftm:	/noise	dmrl has been removed)
91	case Cras:	/noise	line has been disconnected)
92	case Cronhook:	/noise	line has been disconnected)
93	case Crone:	/noise	line has been hung up)
94	golo Bidle:	/noise	(tone has been applied)
515			

FIG. 6  
FROM FIG. 5

FIG. 6

```

1610: switch(op) {
1611:   default:
1612:     goto error;
1613:
1614:   case Cronhook:
1615:     send(x->term, Cldis, 0); // on hook
1616:     send(x->term, Cldis, 0);
1617:     goto idle;
1618:
1619:   case Cibusy:
1620:     send(x->Cstone, Tbusy);
1621:     gotoibusy;
1622:
1623:   case Cring:
1624:
1625: }
```

© 1998 Lucent Technologies Inc.

```

175 } break;
176 } send(x, Ctlone, Tring); //apply audible ringing tone
177 }

178 } Ring;
179 switch (op) {
180 default:
181     goto error;
182 }

183 case Ctrhook:
184     send(x->Term, Cldis, 0);
185     send(x, Cldis, 0);
186     goto Blale;
187 }

188 case Ctlone:
189     goto Bring;
190 }

191 case Ctranswer:
192     break;
193 }

194 goto Bring;
195 }

196 case Ctranswer:
197     break;
198 }

199 send(x, Ctlone, x->Term->lineno); //cross connect the ab
200

```

```

206 ①conn
207 switch(op) {
208 default:
209 case Gconn:
210     goto Bconn;
211
212 case Cronhook:
213     send(x->term, Cdis, 0);
214     send(x, Cdis, 0);
215     goto Bidle;
216
217     goto error;
218
219 */

```

L 10 FIG 7

FROM FIG 6

EIC 7

100

© 1998 Lucent Technologies Inc.

FIG. 8

FROM FIG 7

```

141 send(x, Cldis, 0)
142 goto Bidle;
143 /* one phone has audible ringing
144 goto Bring; Bring: x->state = 7; goto out; Aring;;
145 line 181 sample.2
146
147 switch(op) {
148 default:
149 goto error;
200
201 case Crnhook:
202 send(x->Term, Cldis, 0);
203 send(x, Cldis, 0);
204 goto Bidle;
205
206 case Crone:
207 goto Bring;
208
209 case Canswer:
210 break;
211
212
213 send(x, Ciconn, x->Term->lineno); //cross connect the phones
214 send(x->Term, Ciconn, x->lineno);
215
216 /*
217 * phones are connected
218 */
219 goto Bconn; x->state = 8; goto out; Aconn;;
220 line 204 sample.2
221
222
223 },

```

© 1998 Lucent Technologies Inc.

700

FIG. 9

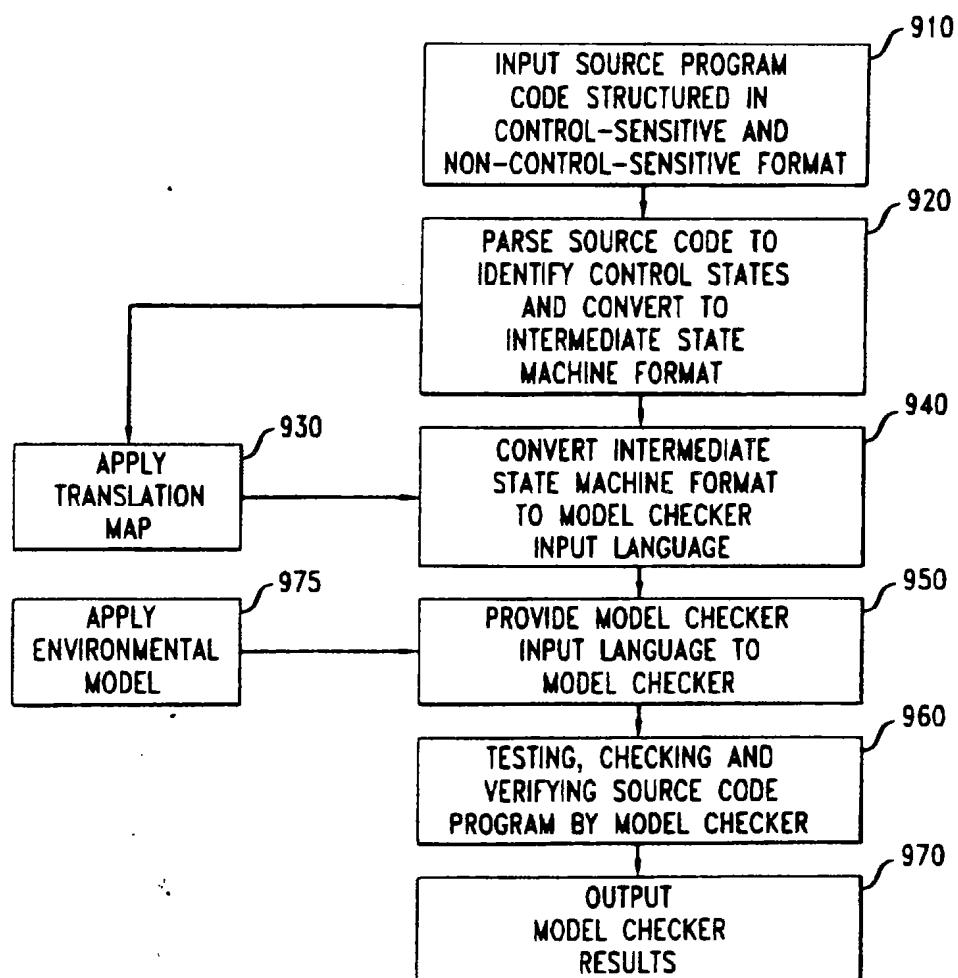


FIG. 10

```

5 %p-template "state_p.lpi"
6 %p-map state.map
7 %event Canswer
8 %event Cbusy
9 %event Cconn
10 %event Cdig1
11 %event Cdig2
12 %event Cdis
13 %event Cdm
14 %event Cfhook
15 %event Chook
16 %event Cprovic
17 %event Cring
18 %event Ctime
19 %event Ctrone
20 %state S_dead
21 %state S_idle
22 %state S_idle_1
23 %state S_idle_2
24 %state S_digits
25 %state S_digits_1
1020 %state S_ring
26 %state S_conn
27 %state S_conn
28 %state S_busy
29 %state S_busy

63 S_digits; onreceipt /* #126 */;
64 (x->time==seconds(6); /* #126 */;
65 else error
66 |Cfhook S_idle
67 send(x,Cdis,0); /* #132 */;
68 |Cfone S_digits_E6
69 |Cdig1 S_digits_E6
70 |Cdig2 S_digits_E6
71 |Ctime S_busy
72 ( send(x,Cdis,0); /* #148 */;
73 send(x,Cfone,0); /* #149 */;
74 ;

1000
1040
1045
1050
1055
1060
1065
1070
1075
1080
1085
1090
1095
1100

102 S_ring : else "error"
103 |Cfhook S_idle
104 |send(x->term,Cdis,0); /* #189 */;
105 |Cfone S_ring
106 |Canswer S_conn
107 |send(x,Cconn,x->term->lineno); /* #200 */;
108 |send(x->term,Cconn,x->lineno); /* #201 */;
109
110

1030
1035
1040
1045
1050
1055
1060
1065
1070
1075
1080
1085
1090
1095
1100

30

```

FIG. 11

```

1# Names of events introduced only here:
2  %event C1conn
3  %event C1dis
4  %event C1dm
5  %event C1ring
6  %event C1answer
7  %event C1busy
8  %event

9# Statements that are not the target of this check:
10# 
11  p=dialup((x->digits, x->ndigits) true /*not target of this check*/
12  x->ndigits=0 true /*ditto*/
13  x->term=0 true
14  x->term=lookup(p) true
15  print(x, C1one, 0) true
16  send(x, C1one, C1busy) true
17  send(x, C1one, C1dm) true
18  send(x, C1one, C1ring) true
19  send(x, C1one, C1answer) true
20  send(x, C1one, C1busy) true
21  send(x, C1one, C1ring) true
22  send(x, C1one, C1stream) true
23# allow these properties to be assumed either true
24# or false (introducing non-determinism in the abstraction):
25# 
26  {(!ndigits==1) true /*assume condition true*/
27  !(x->ndigits==1)) true /* or its negation true */
28# 
29  (p==0) true /*ditto*/
30# ...

```

1110 {

31 !((p==0)) true /\*ditto\*/ 1100

32 !(x->term==0)) true

33 !(x->term==0)) true

34 !(x->term==0)) true

35 Statements that are relevant and directly represented:

36#

37 x->time=seconds(6) Timer = true /\*start timer\*/
38 send(x, C1conn, x->term->lineno) acs1conn /\*send a message\*/
39 send(x, C1dis, b) acs1cdis
40 send(x, C1dm, b) acs1cdm
41 send(x, C1dm, 1) acs1cdm1
42 send(x->term, C1conn, x->lineno) acs1conn
43 send(x->term, C1dis, 0) acs1cdis
44# Nondeterministically abstract in one of two ways:
45#

46 send(x->term, C1ring, 1) if :: acs1Cring; acs1Canswer
47 send(x->term, C1ring, 1) if :: acs1Cbusy

*FIG. 12*

1200

```
1 /*  
2  * source: state */  
3  
139 A_S_digits_Cronhook;  
140 /* { send(x, Ctdis, 0); ** #132 **; } */  
|  
TO FIG 13
```

© 1998 Lucent Technologies Inc.

*FIG. 13*

FROM FIG 12

1200

```
141  goto BS_idle;  
142  :: else ->  
      :  
302  qin'Crprovlc      /*provision the phone*/  
303  }
```

© 1998 Lucent Technologies Inc.

*FIG. 14*

1400

```
1  /*
2  *source: state.x
3  *
152 od
153 }
```

© 1998 Lucent Technologies Inc.

FIG. 15

1500

```

1  /* Spin template */
2
3
19 prototype context ()
20 {   mtype msg;
21
22 end:  do
23   :: Timer && timeout -> /* the timer expires */
24     Timer = false; /* turn it off */
25     qin!Crttime /* notify the switch */
26
27   :: atomic {
28     acs?msg-> /* message sent to environment */
29     Timer = false; /* on arrival of any event: timer cancels */
30
31   if
32   :: msg == Ctdtmf -> /* digit recognition */
33     qin!Crdtmf; /* must acknowledge this event */
34     dialtone = 1 - dialtone /* toggle on-off */
35
36   :: msg == Ctdis ->
37     qin!Crdis; /* acknowledge */
38     dialtone = false /* disconnecting - dialtone off */
39
40   :: msg == Ctring -> /* phone rings */
41     qin!Crring
42
43   :: msg == Ctanswer -> /* phone answered */
44     qin!Cranswer
45
46   :: msg == Ctbusy -> /* phone is busy */
47     qin!Crbusy
48
49   :: msg == Crdis, msg == Ctconn, msg == Crconn ->
50     skip /* messages not yet considered */
51
52   :: else -> /* nothing else should happen */
53     printf("context: cannot happen\n");
54     assert(false) /* assert that we cannot reach this */
55   fi
56 }
57 }
58 od
59
60 od
61
62 od
63
64 od
65
66 od
67
68 od
69
70 od

```

© 1998 Lucent Technologies Inc.

FIG. 16

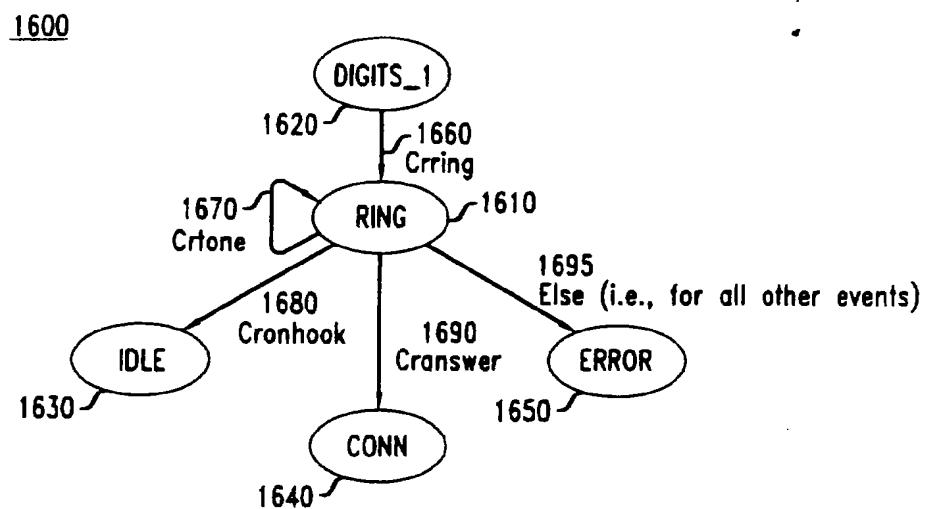


FIG. 17

1700

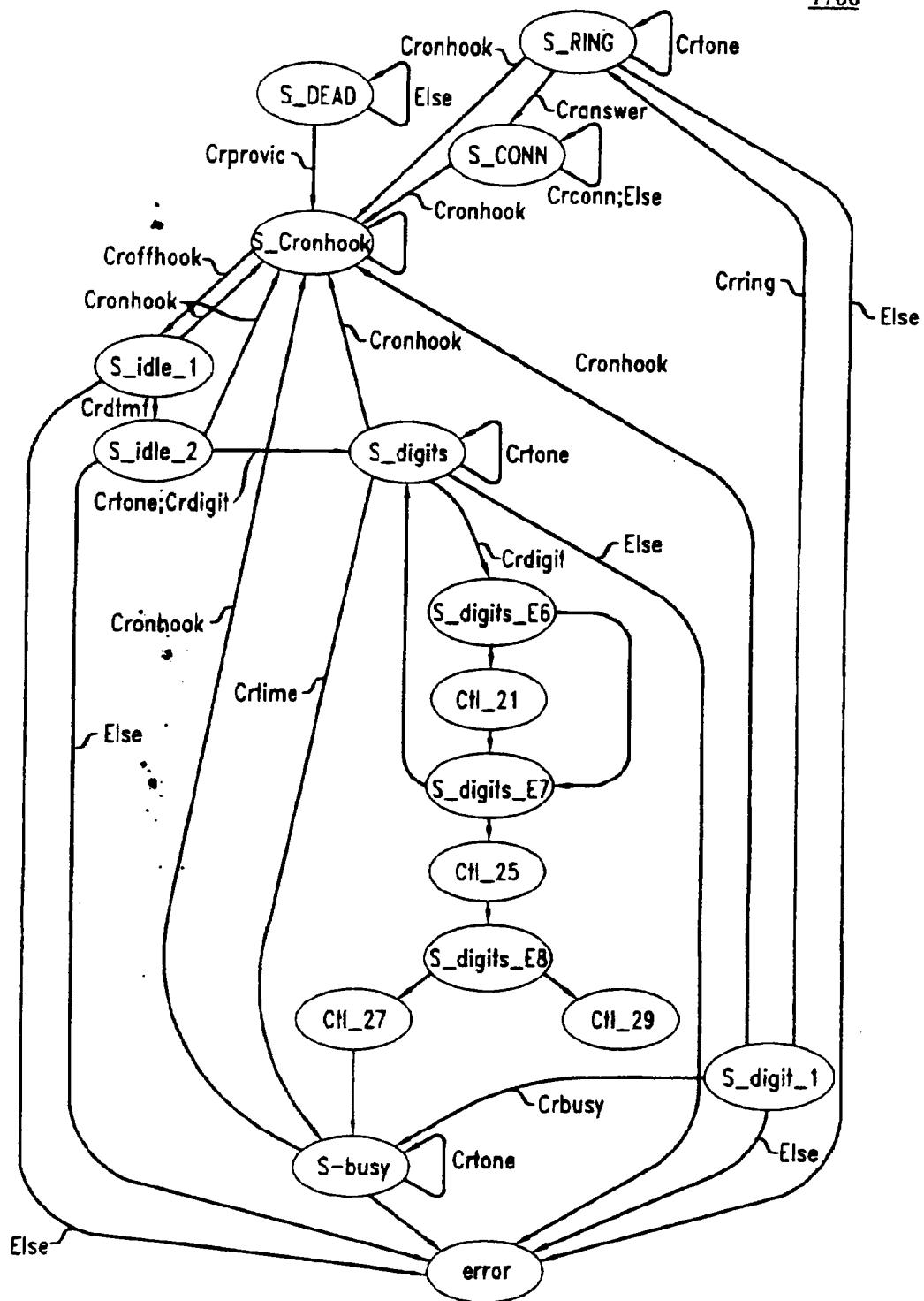


FIG. 18

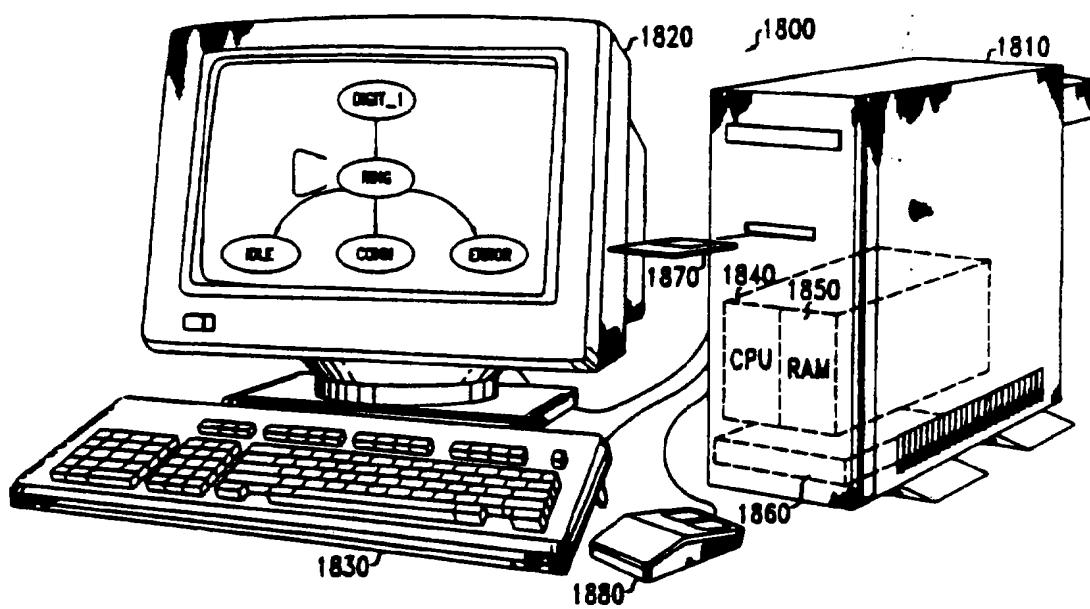


FIG. 19

```

1 Error scenario for specification.          1900
2 Reported by a Spin verification
3
4
5 < state S_dead >          /* control state */
6 Crprovlc                  /* event received */
7
8 {x->term=0; ** #79 **; }
9
10 < state S_idle >          /* C code executed */
11 Croffhook                  /* etc. */
12
13 { x->ndigits=0; ** #99 **; send(x,Cdltmf,1); ** #100 **; }
14
15 < state S_idle_1 >
16 Crdltmf
17
18 { send(x,CHone,Tdial); ** #111 **; }
19
20 < state S_idle_2 >
21 Crdigit
22
23 { x->time=seconds(6); ** #126 **; }
24 p=dialplan(x->digits,x->ndigits); ** 140 **; }
25 ((x->ndigits == 1))
26 ((p == 0))
27 {x->term=lookup(p); **154 **; }
28 {x->term == 0}
29 { send(x,CHone,Treorder); ** #156 **; }
30
31 < state S_busy >          /* unspecified event */
32 Crdigit
33
34 reached error state          1920
35
36 spin: line 239 "state.p" Error: assertion violated
37 spin: trail ends after 79 steps
38 #processes: 5
39     queue 3 (acs):
40         dialtone = 1
41         Timer = 1
42 79: proc 4 (context) line 245 "state.p" (state 26) <valid endstate>
43 79: proc 3 (subscriber) line 288 state.p (state 9)
44 79: proc 2 (state) line 240 state.p (state 189) <valid endstate>
45 79: proc 1 (:int:) line 303 state.p (state 6) <valid endstate>
46 79: proc 0 (talker) line 138 ./state/1 (state 145) <valid endstate>
47 5 processes created

```



European Patent  
Office

## EUROPEAN SEARCH REPORT

Application Number  
EP 99 30 9818

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
X	EP 0 685 792 A (AT&T CORP.) 6 December 1995 (1995-12-06) * column 1, line 35 - line 56 *	1-18	G06F11/00
A	GERARD J. HOLZMANN: "The Model Checker SPIN" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING., vol. 23, no. 5, May 1997 (1997-05), pages 279-295, XP000720371 IEEE INC. NEW YORK., US ISSN: 0098-5589 * page 280, left-hand column, line 21 - right-hand column, line 19 *	1-18	
TECHNICAL FIELDS SEARCHED (Int.Cl.7)			
G06F			
The present search report has been drawn up for all claims			
Place of search	Date of completion of the search	Examiner	
THE HAGUE	3 May 2000	Corremans, G	
CATEGORY OF CITED DOCUMENTS			
X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document			
T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document			

**ANNEX TO THE EUROPEAN SEARCH REPORT  
ON EUROPEAN PATENT APPLICATION NO.**

EP 99 30 9818

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on. The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

03-05-2000

Patent document cited in search report	Publication date		Patent family member(s)	Publication date
EP 685792 A	06-12-1995		CA 2147536 A	02-12-1995
			JP 7334566 A	22-12-1995
			US 5615137 A	25-03-1997